Workload distribution and balancing in FPGAs and CPUs with OpenCL and TBB

Rafael Asenjo[a] , Angeles Navarro[a] , Andres Rodriguez[a] and Jose Nunez-Yanez[b]

[a] *Universidad de Málaga, Andalucía Tech, Spain*
[b] *University of Bristol, UK*

Abstract. In this paper we evaluate the performance and energy effectiveness of FPGA and CPU devices for a kind of parallel computing applications in which the workload can be distributed in a way that enables simultaneous computing in addition to simple off loading. The FPGA device is programmed via OpenCL using the recent availability of commercial tools and hardware while Threading Building Blocks (TBB) is used to orchestrate the load distribution and balancing between FPGA and the multicore CPU. We focus on streaming applications that can be implemented as a pipeline of stages. We present an approach that allows the user to specify the mapping of the pipeline stages to the devices (FPGA, GPU or CPU) and the number of active threads. Using as a case study a real streaming application, we evaluate how these parameters affect the performance and energy efficiency using as reference a heterogeneous system that includes four different types of computational resources: a quad-core Intel Haswell CPU, an embedded Intel HD6000 GPU, a discrete NVIDIA GPU and an Altera FPGA.

Keywords. FPGA, OpenCL, heterogeneous scheduling, streaming application.

Introduction

Energy efficiency is a fundamental consideration in both embedded and High Performance Computing (HPC). In HPC computational complexity means that power requirements of server racks have reached the level of megawatts and electricity bills continue to increase. Additionally, device power density is limiting the option of using more logic to solve problems in parallel. Confronted with this energy challenge designers are moving towards heterogeneous architectures in which specialized hardware units accelerate complex tasks. A good example of this trend is the introduction of GPUs (Graphics Processing Units) for general purpose computing with the help of parallel programming framework such as OpenCL (Open Computing Language). OpenCL is a cross-platform parallel programming model designed to facilitate effective use of heterogeneous processing platforms. FPGAs (Field Programmable Gate Arrays) are an alternative high performance technology that offers bit-level parallel computing in contrast with the word-level parallelism deployed in GPUs and CPUs. Bit-level parallel computing fits certain algorithms that cannot be parallelized easily with traditional methods. FPGAs also excel in tasks that require very low latency such as financial computing thanks to its ability to establish direct wired connections between streaming processing pipelines customized for the input data

without intermediate memory transfers. Recently, FPGAs manufacturers have been pioneering OpenCL for FPGAs aiming to overcome their low-level programming models. In this paper we consider an accelerated system that combines FPGAs and GPUs with OpenCL support together with a multi-core CPU that can act as host or actively participate in the computation. We evaluate different approaches to distribute the workload between host and accelerator in order to exploit processing threads mapped to the host in addition to the accelerator. The objective is to measure if simultaneous computing among these devices could be more favorable from and energy and/or performance points of view compared with off-loading and CPU idling. As a case study, we introduce ViVid, an application that implements an object (e.g., face) detection algorithm [1] using a "sliding window object detection" approach [2]. ViVid consists of 5 pipeline stages as we depict in Fig. 1. The first and the last one are the Input and Output stages (serial), whereas the three middle stages are parallel (or stateless).

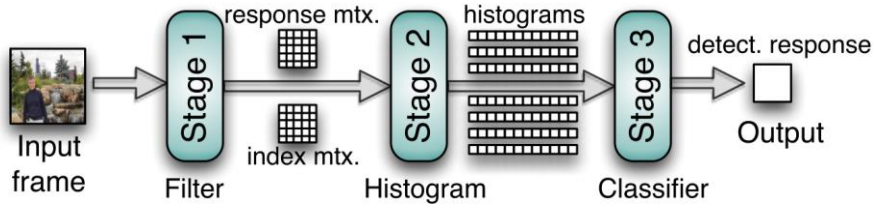

Figure 1. Vivid application

The main contribution of this paper is the study of the performance portability that a realistic OpenCL application can obtain in FPGA devices compared with GPU and CPU devices. We also propose a mechanism to simultaneously compute the OpenCL kernels with FPGA (GPU) and a multicore CPU, and explore its effects on performance and energy.

The rest of the paper is organized as follows. Section 1 reviews background and related work that uses FPGAs with OpenCL for processing intensive applications. Section 2 presents details of our test system that also considers discrete and embedded GPU devices as comparison points. Section 3 evaluates the performance, power and energy characteristics of each of these devices individually to asses their suitability. Section 4 explores the possibility of using the FPGA or GPU device simultaneously with the multicore CPU to further accelerate the system and reduce overall processing time. Section 5 presents the conclusions and proposes future work.

1. Background and Related work

There are significant efforts at using FPGAs as an acceleration technology based on the same programming model as the one used for other accelerators in the system. One of the objectives is to make FPGAs more attractive devices for software engineers by raising the level of abstraction and significantly increasing the level of productivity [3]. FPGAs are hardware configurable after manufacturing so custom circuits can be

created to match the application requirements. Modern FPGAs are formed by a 2-dimensional array of basic logic cells that are used to form logic and also include additional embedded memory, DSP blocks, high speed transceivers and I/Os, CPU cores (e.g. ARM/Intel) and routing which wire together all these elements to create custom circuits. FPGAs have been traditionally programmed using a RTL flow (e.g. VHDL/Verilog) but this has started to change over the last few years. Altera, for example, has introduced the Software Development Kit (SDK) for OpenCL (AOCL) which is an OpenCL-based heterogeneous parallel programming environment for their devices. There are two steps to program a FPGA with an OpenCL application using the AOCL tool. Firstly, the Altera Offline Compiler (AOC) compiles the OpenCL kernels and secondly, the host-side C compiler compiles the host application and then links the compiled OpenCL kernels to it [4]. Each OpenCL kernel is compiled to a custom pipelined-style parallel circuit consuming FPGA resources. AOCL favours a single but very deep pipeline in which multiple work-items execute in parallel. It is possible to replicate pipelines for even more parallelism creating multiple compute units. The custom FPGAs circuits can provide better performance per watt compared to GPUs for certain applications [5] according to the Altera literature.

The more common approach of using these devices consists of offloading complex kernels to the accelerator by selecting the best execution resource at compile time. It is also possible to do run-time selection by including different versions of the same kernel as long as enough hardware resources are available to implement all the kernel versions simultaneously. The idea of run-time has been explored previously in the literature mainly around systems that combine GPUs and CPUs. For example, selection for performance with desktop CPUs and GPUs has been done in [6] that assigns percentages of work to both targets before making a selection based on heuristics. Energy aware decisions also involving CPUs and GPUs have been considered in [7] which requires proprietary code. A more related work in the context of streaming applications [8] considers performance and energy when looking for the optimal mapping of pipeline stages to CPU and on-chip GPU, although FPGAs are not studied. Recently, IBM has proposed a new programming language called Lime and the Liquid Metal [9] system that combines GPUs, FPGAs and CPUs. This approach uses a proprietary language and cannot leverage the extended efforts invested in creating high-level compilers by the different vendors. SnuCL also proposes [10] an OpenCL framework for heterogeneous CPU/GPU clusters, considering how to combine clusters with different GPU and CPU hardware under a single OS image. In this paper we extend previous research around the idea of run-time selection by considering the efficiency of OpenCL-supported FPGAs compared with GPUs and CPUs and also the possibility of deploying CPUs and FPGAs simultaneously.

2. Test system specifications

In order to perform a fair comparison a single test system is considered in which all the computational resources are installed. This test system is formed by a desktop environment running Windows 7 64-bit and equipped with the different OpenCL targets which are the Terasic DE5-net board that includes an Altera Stratix V A7 FPGA, a Core i7 4770k 3.5 GHz Haswell CPU with a HD6000 embedded GPU and a Nvidia Quadro K600 GPU. We have selected the K600 GPU as an example of low-

power device with comparable thermal power dissipations rated at 25 W for the FPGA and 40 W for the GPU.

Currently the larger FPGA available in the Altera OpenCL program is the Stratix V D8. Our device is the Stratrix V A7 FPGA that is comparable in size to the D8 in all the parameters except in embedded DSP blocks. The Altera Stratrix V A7 FPGA contains 622,000 logic elements comparable to the 695,000 available in the Stratix V D8 FPGA. The embedded memory in both FPGAs is rated at 52 Mbit for both devices, which can be used as private or local memory in OpenCL. The main difference is that the A7 device contains 256 27x27 DSP blocks, which is much lower than the 1963 DSP blocks present in the D8. The device memory is formed by 4GB DDR3 at 800 MHz installed in the DE5-net board with a max bandwidth of 25 GB/s.

The Nvidia Quadro K600 has 192 CUDA cores, 1 GB DDR3 memory, the max bandwidth is equivalent to the DE5-net and the top floating point performance is indicated as 336 GFLOPs. This top performance is much more difficult to determine in the FPGA device since the amount of floating point hardware that can be implemented depends on the type of operations and the logic that is required by the rest of the algorithm. One of the strengths of the FPGA is its abundant and flexible local memory that can be used to buffer data reducing the need for off-chip memory accesses. The K600 and the Stratix V are both manufactured using a 28 nm process and were introduced in 2012 and 2011 respectively.

Table 1 shows the measurements taken to evaluate the (idle) power of the system when no benchmarks are running and only OS background activity is present. With this data it is possible to estimate that the idle power of the GPU card is around 8 Watts while the idle power of the FPGA card is approximately 22 Watts. Note that this is the power of the whole card including device memory and not just the GPU or FPGA devices. The system without accelerators shows the idle power without any cards installed. As a reference the idle power of the CPU socket that includes the quad-core and the embedded GPU has been measured at 9 Watts using the Intel energy counters available on chip. We can consider these values estimations that should be further refine with proper instrumentation in future work.

Table 1. Analysis of idle power for whole system

	Idle Power (Watts)
No accelerator	44
GPU in	52
FPGA in	66
GPU-FPGA in	74

3. Performance and energy evaluation of individual devices

Table 2 shows the results of running the Vivid application accelerated by a single device for the configuration that includes the three processing kernels that correspond to the parallel stages: filter (F1), histogram (F2) and classifier (F3). A stream of 100 LD (416x600) images was used in our experiments. The power has been estimated by

measuring the wall plug power with the benchmark running and then subtracting the idle power reported in section 3. The host code, accelerator code for the CPUs and accelerator code for the GPUs have been compiled with Visual studio 2013, TBB 4.3 and Nvidia OpenCL 6.0 respectively. The GPUs' OpenCL kernels were compiled using -03. The FPGA code is compiled offline with version 14.1 of the Altera AOC tools with the addition of the -fpc pragma that instructs the compiler to remove floating-point rounding operations and conversions whenever possible and carry additional mantissa bits to maintain precision during internal operations. The utilization ratios of the different logic resources and resulting kernel frequency for the FPGA are shown in Table 3 depending in which stage is implemented in the device.

Table 2. Power and performance analysis

Device	Time (sec.)	Throughput (fps)	Power (Watts)	Energy (Joules)
CPU (1 core)	167.610	0.596	37	6201
CPU (4 cores)	50.310	1.987	92	4628
FPGA	13.480	7.418	5	67
On-chip GPU	7.855	12.730	16	125
Discrete GPU	13.941	7.173	47	655

Table 3. FPGA kernels resource utilization and frequency

Kernel	Logic resources (Altera Logic Modules)	DSP blocks	Memory (bits)	Freq (Mhz)
F1	77,425 (32%)	44/256	6,896,354 (13%)	283
F3	163,733 (69%)	106/256	18,693,257 (35%)	225
F1+F2+F3	209,848 (89%)	162/256	24,679,220 (47%)	186

Table 2 shows that the power used by the FPGA card when it is active is significantly lower than the other resources. This is surprising since its TDP is comparable to the discrete GPU. An analysis of the profiling reports shows that the FPGA pipeline stalls waiting for memory around 10% of the time which could have an implication in power but in any case this result indicates a good power efficiency of the FPGA card with this application.

Fig. 2 shows the breakdown of the average time per frame (in ms.) for each stage of ViVid and on each device. Labels onGPU and disGPU represent de on-chip and discrete GPUs, respectively. Interestingly, stages 1 and 2 map better on the on-chip GPU, whereas stage 3 behaves best on the FPGA. We also show the times that data transfers (host-to-device and device-to-host) require in each case. Clearly, these times are lower for the integrated GPU because it does not require PCIe bus transactions, as the others devices do. For instance, data transfer times on the FPGA are 1.7x higher than those on the on-chip GPU. Fig. 2 suggest that the most effective configuration should combine the embedded GPU and FPGA simultanously.

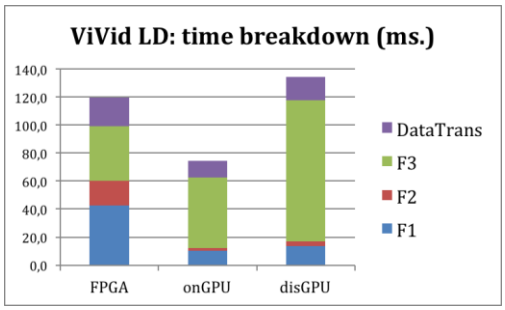

Figure 2. Breakdown of times for each stage and on each device.

4. Simultaneous computing experimental results

When an application like ViVid runs on a heterogeneous architecture, it is possible to distribute the workload so that more than one device is active simultaneously. In this case many possible configurations are possible. In this section we focus on combining CPU and GPU or FPGA devices and leave an extension that uses all the accelerator resources simultaneously (i.e. CPU and GPU and FPGA) as future work. Fig. 3 graphically depicts the possible mappings for the three parallel stages to the CPU cores and the accelerator (GPU or FPGA). In addition, it is possible to control the number of active CPU cores that compute together with the accelerator to minimize the execution time, the energy consumption, or both (depending on the metric of interest). The pipeline mapping determines the device where the different stages of the pipeline can execute. Let's assume that a pipeline consists of S1, S2, ... Sn stages. We use a n-tuple to specify a static stage mapping to the accelerator and the CPU devices: {m1, m2, ..., mn}. The i-th element of the tuple, mi, will specify if stage Si can be mapped to the accelerator and CPU, (mi= 1), or if it can only be mapped to the CPU (mi= 0). If mi= 1, the item that enters stage Si will check if the accelerator is available, in which case it will execute on it; otherwise, it will execute on the CPU. For instance, for the ViVid example of Fig. 3 we represent the tuples (row major order): {1,1,1}, {1,0,0}, {0,1,0}, {0,0,1}, {1,1,0}, {1,0,1}, {0,1,1}, {0,0,0}. In our implementation, mapping {1,1,1} represents a special case: if the accelerator is available when a new item enters the pipeline, then all the stages that process it will be mapped to it.

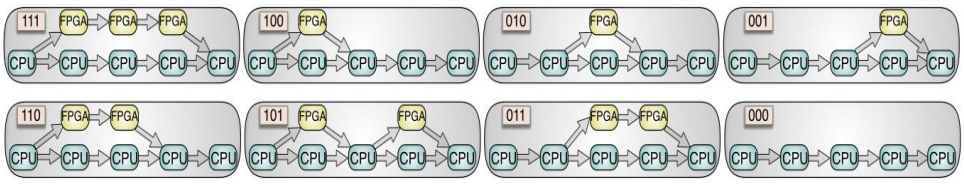

Figure 3. Possible stage mappings of stages to the accelerator and/or CPU for ViVid.

Fig. 4 shows the performance (fps or frames per second) and energy usage (Joules) after using configuration {100} that maps the first Vivid filter (F1) to the accelerator. The x-axis represents the number of threads that goes from 1 to 8, to evaluate the hiper-threading feature of the Haswell CPU. Note that the energy is represented in log scale. In the figure we see that if the number of threads is limited to one then only the FPGA or GPU accelerator are used, but if the number of threads is higher the scheduler uses the cores as additional computing resources. It is possible to observe in Fig. 4 that when only one thread is active the performance of all the accelerators is comparable but the FPGA configuration is much more energy efficient. In this case, the FPGA energy consumption is 11% and 30% more energy efficient than the consumption on the integrated and discrete GPUs, respectively. Adding additional threads results in the cores participating in the computation, which increases throughput but it is more energy inefficient. When the third stage (F3) is mapped as shown in Fig. 5 both energy efficiency and throughput increase when both CPUs and accelerators participate in the computation. The combination of CPU and FPGA is the best choice for this stage for both energy and performance purposes. Now with 8 threads, the FPGA is 39% faster and 54% more energy efficient than the discrete GPU.

Finally, Fig. 6 shows the case when all the stages try to use the accelerator as its first choice. This configuration results in the higher performance achieving almost 20 fps for the embedded GPU case, 12 fps for the FPGA and 8 fps for the discrete GPU. For a single thread, performance is comparable for all the accelerators but the FPGA is the most energy efficient. The FPGA is 1.8x and 9.2x more energy efficient than the embedded and the discrete GPUs, respectively. Once the number of threads increases and the cores participate in the computation, the combination of the embedded GPU and the CPU is more energy efficient than the FPGA and CPU, but overall energy efficiency decreases because we reach the point of diminishing returns when using 4 threads. Critically, this reduction of energy efficiency takes place despite the overall increase in throughput. The fact that the embedded GPU is more energy efficient than the FPGA for the multi-threaded configuration could be explained in part due to the overheads of the data movements over the PCIe bus compared with the tighter integration between CPUs and embedded GPUs that share the L3 cache. In the case of the discrete accelerators the presence of the PCIe bus can quickly become a performance bottleneck especially when computation is shared between cores and accelerators. Other factor that explains the higher performance of the integrated GPU is that the OpenCL compiler generates a very efficient binary for the first kernel/stage, as we saw in Fig. 2.

In any case further work is necessary to understand how the tighter integration results in better performance and energy efficiency. Overall, the discrete GPU results cannot match the results observed with the embedded GPU and FPGA configurations, mainly due to the less efficient third kernel (see Fig. 2). Let's keep in mind that we use the same OpenCL source code of the kernels for all accelerators. This was done on purpose to evaluate performance portability of the same code version on the different devices. As a future work we plan to study which OpenCL optimizations work well for each accelerator.

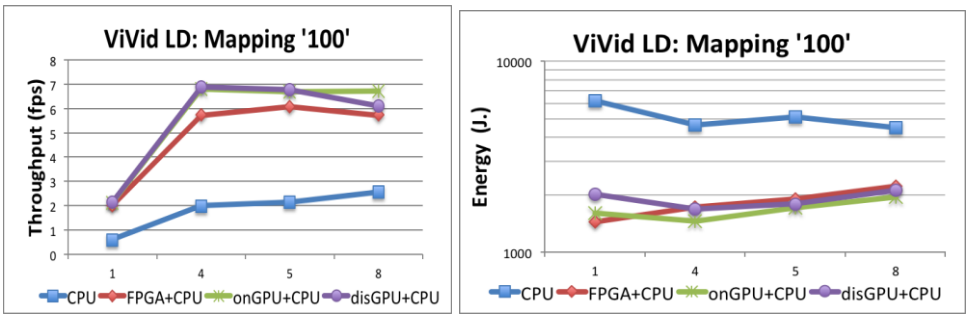

Figure 4. Stage 1 acceleration with different hardware configurations.

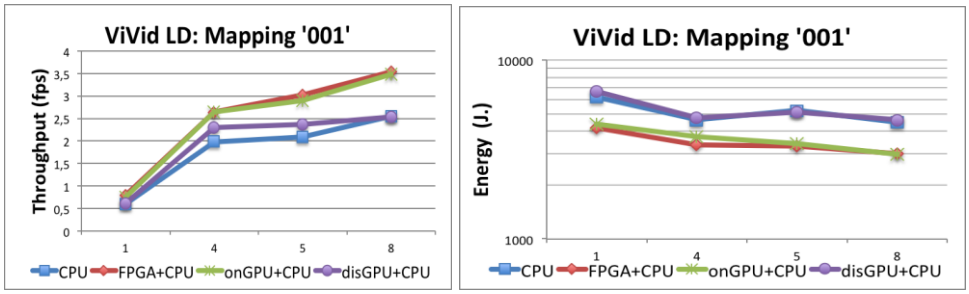

Figure 5. Stage 3 acceleration with different hardware configurations

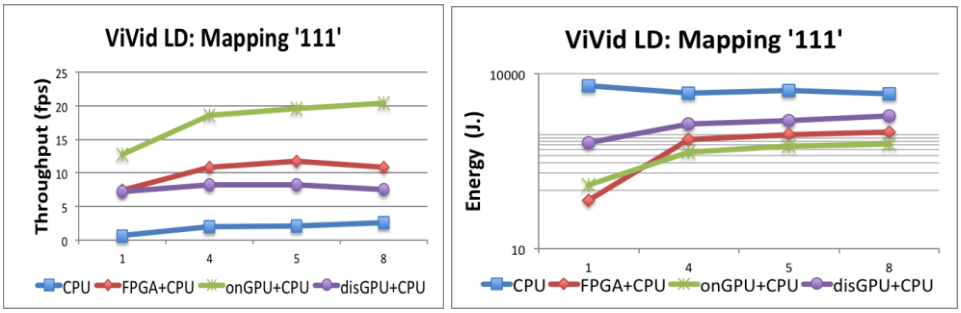

Figure 6. Stages 1+2+3 acceleration with different hardware configurations.

5. Conclusions

This paper has investigated the effects of distributing the workload between accelerators consisting of GPUs, FPGAs and CPU cores. From the conducted experiments a number of conclusions can be drawn. The first conclusion is that the FPGA device programmed with OpenCL is competitive with other accelerator types and shows good performance portability characteristics. It obtains the most energy efficient configuration superior to both embedded and discrete GPUs as seen in Fig. 6.

This takes place despite that the OpenCL code has not been optimized for the FPGA device and is essentially the same code used to program the GPU devices. The second conclusion is that the single thread configurations in which the CPU cores do not participate in the computation are more energy efficient. On the other hand Fig. 6 also shows that if the objective is to obtain raw performance the additional deployment of the cores is beneficial and it can double throughput. Future work involves optimizing the code for the FPGA device focusing on reducing the number of pixels that must be fetched from device memory by reusing pixels shared by the filter operations and also increasing hardware occupancy by activating all kernels simultaneously instead of serially as done in these experiments. We also plan to develop a system that will allow to use the GPU and FPGA cores simultaneously with the CPU cores, deploying all the computation resources at the same time.

Acknowledgments

This work was partly founded by Universidad de Málaga:"Ayudas para el fomento de la internacionalización" funded by ERDF and by the following Spanish projects: TIN 2013-42253-P and P11-TIC-08144. We also aknowledge the support from the EPSRC, UK to perform this research thanks to the ENPOWER EP/L00321X/1 award.

References

[1] AMert Dikmen, Derek Hoiem, and Thomas S Huang. A data driven method for feature transformation. In Proc. of CVPR, pages 3314{3321, 2012.

[2] Michael Jones and Paul Viola. Fast multi-view face detection. Mitsubishi Electric Research Lab TR-20003-96, 3, 2003.

[3] Czajkowski, et. Al. "From opencl to high-performance hardware on FPGAS," Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on , vol., no., pp.531,534, 29-31 Aug. 2012

[4] Altera SDK for OpenCL Programming Guide, available at: http://www.altera.co.uk/literature/hb/opencl-sdk/aocl_programming_guide.pdf

[5] OpenCL on FPGAs for GPU Programmers, available at: http://www.altera.co.uk/products/ software/opencl/opencl-index.html

[6] Prasanna Pandit and R. Govindarajan. 2014. Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices. Annual IEEE/ACM International Symposium on Code Generation and Optimization.

[7] Dolbeau, R.; Bodin, F.; de Verdiere, G.C., "One OpenCL to rule them all?," Multi-/Many-core Computing Systems (MuCoCoS), 2013 IEEE 6th International Workshop on , vol., no., pp.1,6, 7-7 Sept. 2013

[8] A. Vilches, A. Navarro, R. Asenjo, F. Corbera, R. Gran, and M. Garzaran *Mapping streaming applications on commodity multi-CPU and GPU on-chip processors*, IEEE Tranactions on Parallel and Distributed Systems, DOI: 10.1109/TPDS.2015.2432809, May 2015.

[9] Takizawa, H.; Sato, K.; Kobayashi, H., "SPRAT: Runtime processor selection for energy-aware computing," Cluster Computing, 2008 IEEE International Conference on , vol., no., pp.386,393, Sept. 29 2008-Oct. 1 2008

[10] Joshua Auerbach, et. Al. "A compiler and runtime for heterogeneous computing". In Proceedings of the 49th Annual Design Automation Conference (DAC '12). ACM, New York, NY, USA, 271-276.